

Hourglass Automata

Yuki Osada, Tim French, Mark Reynolds, and Harry Smallbone

The University of Western Australia.

yuki.osada@research.uwa.edu.au, {tim.french,mark.reynolds}@uwa.edu.au, 21306592@student.uwa.edu.au

In this paper, we define the class of *hourglass automata*, which are timed automata with bounded clocks that can be made to progress backwards as well as forwards at a constant rate. We then introduce a new clock update for timed automata that allows hourglass automata to be expressed. This allows us to show that language emptiness remains decidable with this update when the number of clocks is two or less. This is done by showing that we can construct a finite untimed graph using clock regions from any timed automaton that use this new update.

1 Introduction

Hybrid systems, as defined by Alur et al. [6], consist of finite automata where the transitions between states are guarded by a set of variables that change over time. Timed automata are a subset of hybrid systems that have been studied extensively due to their usefulness in industrial modelling. Timed automata as shown in Alur and Dill [3] model time using a finite set of monotonically increasing real-valued clocks. The clocks can be used in guards in automaton transitions and have a number of operations associated with them. They can be reset to 0 independently of each other. Cassez and Larsen [7] also define a class of timed automata, stopwatch automata, which allows the clocks to be stopped and started, resulting in an expressive power equal to linear hybrid automata. Stopping time generally causes undecidability, but Bérard et al. [11] show it is possible to introduce a restricted form of stopping time with the interrupt timed automata, where clock levels are used to allow values that a clock can be compared to be determined during execution.

Alur and Dill prove the emptiness problem for timed automata with integer clock guards to be decidable. By creating a clock region automaton, state reachability is PSPACE-complete, leading to formal method applications such as UPPAAL [8]. UPPAAL is a model-checking tool that uses symbolic representations of integer clocks to verify timed automata models.

The decidability of language emptiness for several types of timed automata updates have been looked at Bouyer et al. [9]. They show that the updates $x := c$, $x := y$ are decidable, and $x := x + 1$, $x := y + c$ are decidable for 3 or more clocks only when no diagonal constraints are used. This shows that we can change decidability results by removing diagonal constraints. Further, Brihaye et al. [10] show that under bounded time horizons, reachability of linear hybrid automata can be improved.

We introduce a new update operation to the clocks $x := c_x - x$ over the bounded time range $[0, c_x]$, which allow clocks to simulate the behaviour of going backwards while actually moving forwards within a bounded time range. The motivation for this operation can be explained simply in terms of the hourglass problem.

The hourglass problem [4] describes a situation of some finite set of hourglasses which must be used in conjunction to measure a period T . The clocks have co-prime maximum times with T , and can increase or decrease in a range $[0, c]$. Once a clock reaches the bounds of its range, it stops progressing until a flip is made. We define a flip to be a change in the clock's rate of change from increasing to decreasing or

vice versa. Additionally, clocks may only have comparisons to its range bounds. To solve the problem we have two optimisation concerns: firstly, the total amount of time passed in the system, and secondly, the number of flips used to measure T .

Consider an example of timing an egg. The egg must be boiled for exactly 15 minutes with only a 7- and an 11-hourglass timers. This hourglass problem has a number of different solutions, including for example in the two clock problem solving the Bezout identity $ax - by = 1$, where 1 is the greatest common divisor of the co-prime clocks. Applying this solution to the example gives $11x - 7y = 1$, with solution $x = 2$, $y = 3$ as the number of flips for each hourglass, leading to a total of 36 minutes with 5 flips to boil the egg.

The hourglass problem is partially reducible to a timed automaton. By modelling the problem using UPPAAL without the ability to flip the clocks, we obtain a solution of 22 minutes. In this solution, the 7 and 11 clock are both started at $t = 0$: at 7 minutes, the egg is set to boiling with 4 minutes remaining on the 11-clock. The 11-clock is reset at $t = 11$ and the egg continues to be boiled for the following 11 minutes to give $4 + 11 = 15$. However, there is a solution that is even faster than those that can be discovered using timed automata. The optimum solution starts with the egg boiling at $t = 0$ for both clocks. At $t = 7$, the 7-clock is reset and times 4 minutes until $t = 11$. Now we use the new operation to flip the hourglass such that it times from 4 to 0, giving a total time of 15 minutes with 2 hourglass flips. This solution cannot be found using monotonically increasing clocks. Introducing the new operation has useful applications in UPPAAL and, by extension, industrial model-checking.

2 Applications

The hourglass analogy is interesting from a problem solving aspect, but it is also very relevant in the context of industrial applications of hybrid automata. There are some key features of hourglass automata to characterize the problems we are interested in:

- Hourglasses can represent time only up to a fixed maximum time. Once the sand runs out of the hourglass the state of the clock cannot change¹.
- Hourglasses cannot be compared directly to one another. Typically there is no way to tell whether one hourglass has more time remaining than another, (other than coarse estimations based on appearance).
- Hourglasses can be flipped. Flipping an hourglass results in time running backwards, or an accumulated resource running out.

These properties can be found in numerous modelling tasks, particularly those associated with material flow networks. One can imagine that an hourglass represents a silo storing some commodity. A complex process can allow that commodity to accumulate for a time and then to be consumed. Direct applications of this include transportation network with various accumulation points (such as stockpiles or silos at a port). Silos have a fixed maximum, their current content cannot be easily determined, and the time to empty a silo is proportional to (or at least related to) the time taken to fill the silo. We are also interested in modelling more complex systems such as trucks moving minerals about a mine site. Trucks can act as mobile silos, but also a fragment of the process, such as the trucks fuel tank, is analogous to an hourglass. In many of these cases, we would expect the time to fill a silo, or truck, or fuel tank, to be proportional (though not equal) to the time to empty it. In this paper, we only consider clocks which allow time

¹This is reminiscent of the normalisation process used in [3].

to change at rates of 1 or -1 (i.e. silos fill up and empty at the same rate) to establish the theory. In future work we will investigate systems without this restriction. This extension would be similar to the multi-rate automata considered in [5].

3 Hourglass Automata

3.1 Preliminaries

Timed automata are finite automata coupled with a finite set of real-valued variables, called *clocks*, which all increase at unit rate at all locations. Each of these clocks may be reset to zero immediately after a transition, and these transitions can have clock constraints associated to them. The clock constraints on transitions are called *guards*, and unless the constraints in the guard are met, the transition cannot be taken. Similarly, constraints on locations are called *invariants*, and the constraint must be met to stay in that location. The location invariants and transition guards may only consist of comparisons between clock values and non-negative integer constants. We restrict it to integer values since any rational constant can be multiplied by a large enough value that they all become integers.

A *clock valuation* is a map from clock variables to non-negative real-values. States in timed automata are written as location and clock valuation pairs, (s, v) . Since there can be an infinite number of clock valuations, we can have an infinite number of states in timed automata. There are two types of state transitions in a timed automaton:

- a *delay* transition $(s, v) \rightarrow_d (s, v + t)$ for some $t \geq 0$, where $v + t$ is the clock valuation v with all clock values incremented by t .
- an *action* transition $(s, v) \rightarrow_a (s', v')$ for some action a , where s' is a location directly connected to s in the automaton and v' is the valuation v after the transition updates are applied.

The only *clock update* available in the standard timed automata is the reset operation, $x := 0$.

For every clock $x \in X$, we will let c_x be the largest integer constant that x is compared against in the clock constraints. Once x exceeds c_x , the valuation of x can be considered to be ∞ as its value would be indistinguishable from any value above c_x . While the values are still distinguishable, we can split the fractional and integral components as comparisons in clock constraints are all with integers. So for some $t \in \mathbb{R}^+$, let $fr(t)$ be the fractional component of t , and $\lfloor t \rfloor$ be the integral component of t . This gives us $t = \lfloor t \rfloor + fr(t)$.

Hourglass automata are extensions of timed automata that can have clocks that go backwards as well as forwards, and these clocks are bound to the range $[0, c_x]$. Additionally, the guards and invariants in a hourglass automata are limited to comparing a clock x to its bounds: 0 and c_x . Comparing clocks or comparing clocks to other constants does not make sense for hourglasses. Additionally, clocks in an hourglass automata can potentially be stopped to simulate the process of placing an hourglass on its side.

3.2 Syntax

We first introduce clocks that behave like hourglasses with clocks that are bound to a range.

DEFINITION 1 An *hourglass clock* x is a clock that can have a value in $[0, c_x]$, and rate of change in $\{-1, 0, 1\}$. The value of a clock cannot exceed c_x nor go below 0, so the clock stops progressing past those points.

We will now define the main feature of hourglasses, which is their ability to be flipped, and have the clock progress backwards.

DEFINITION 2 A *flip* of a clock is an operation that multiplies the rate of change of the real-value of the clock by -1 . All clocks initially have a rate of change of 1, so the flip operation alternates the rate of change between 1 and -1 .

Another feature of hourglasses is that time can be stopped by placing them on their side.

DEFINITION 3 Hourglass clocks can be stopped, and then started again. This simulates the process of placing a hourglass on its side, and later placing them back upright again. We will call the operation that switches the clock's state between these two as *toggling* the clock.

The rate of change of a clock can be tracked by extending the definition of a timed automata state to include $d \in D : X \rightarrow \{-1, 1\}$ that maps clocks $x \in X$ to a value in the set $\{-1, 1\}$, which encodes the direction time is progressing for a clock. This multiplies the number of states by $2^{|X|}$, where $|X|$ is the number of clocks in the system. We can further extend this to record the stopped clocks by mapping clocks to a value in $\{-1, -0, 0, 1\}$, resulting in a $4^{|X|}$ multiplier. The -0 is needed to store the direction time should progress when toggled back on. Additionally, flipping a clock when stopped, should swap the direction that time progresses in when the clock is started again.

With the above two definitions, we can now define our hourglass automata.

DEFINITION 4 An *hourglass automaton* is a 7-tuple $A = (\Sigma, S, S_i, S_f, X, I, T)$ such that:

- Σ is a finite set of actions,
- S is a finite set of locations,
- $S_i \subset S$ is a set of initial locations,
- $S_f \subset S$ is a set of final locations,
- X is a finite set of hourglass clocks,
- $I : S \rightarrow C(X)$ is a mapping from locations to clock constraints (the *location invariants*), where a clock constraint $\phi \in C(X)$ maps a clock $x \in X$ to a constraint such that $\phi(x) = v(x) \prec c$ or $\phi(x) = c \prec v(x)$ or $\phi = \phi_1 \wedge \phi_2$, where $c \in \{0, c_x\}$, $\prec \in \{<, \leq\}$, $\phi_1, \phi_2 \in C(X)$, and $v : X \rightarrow \mathbb{R}$ maps clocks to their clock valuation.
- $T \subseteq S \times \Sigma \times C(X) \times 2^X \times 2^X \times S$ is a set of *transitions*, where the 6-tuple $\langle s, a, \phi, \mu_{flip}, \mu_{toggle}, s' \rangle$ is a transition from location s to location s' with the label a . This transition is enabled when the constraint ϕ is met, and taking this transition will flip a set of clocks $\mu_{flip} \subseteq X$, and toggle the progress of time in a set of clocks $\mu_{toggle} \subseteq X$. Note that the standard clock reset $x := 0$ is not available in hourglass automata.

3.3 Timed and Untimed Languages

The *initial states* of an automaton are the elements of $S_i \times v_0$, where $v_0(x) = 0$ for all $x \in X$. The *final states* of an automaton are states that are in the *final locations*. A run of a timed automaton is a sequence of *timed moves*, which represents a delay transition followed by an action transition, from an initial state. A finite run is said to be *accepting* if the run ends in a final state. For every finite run, we have a finite *timed word* that is a sequence of timed moves that represent the run. We then say that the *timed language* accepted by a timed automaton is the set of timed words that are accepting runs.

The *language emptiness* problem is the problem of determining whether the language accepted by an automaton is empty or not. If there is a finite number of states, then this problem is reduced to a graph problem, where we check whether there exists a path from an initial state node to a final state node. This

makes the problem decidable. In timed automata, there can be an infinite number of states of the form $(s, v) \in S \times V$, where $|S|$ is finite but $|V|$ can be infinite. Alur and Dill [1, 3] proved the decidability of the language emptiness problem on timed automata by showing that they can construct a Büchi automaton from a timed automaton, and this Büchi automaton accepts exactly the set of *untimed words* that are equivalent to the timed words accepted by the timed automaton. Instead of exact clock valuations, these untimed words use equivalence classes of clock valuations, called *clock regions*.

4 Hourglass-Flip Expressible Update for Timed Automata

We showed earlier that the states in a hourglass timed automaton can be represented by the addition of an extra map d to represent each clock's direction (and whether they are stopped or not), so we now introduce a new clock update that allows us to keep a positive time progression like in the standard timed automata.

LEMMA 5 The update $x := c - x$, where $x \in X$ and $c \in \mathbb{Z}_{\leq c_x}$, with the reset operation $x := 0$ on timed automata is capable of expressing the flip operation with bounded clocks from the hourglass automata.

Proof. The hourglass automata only allows comparisons of clocks to the constants 0 and c_x , so we only care about how much time there is until the clock value reaches these two end points.

Let $0 \leq x \leq c_x$, then $a_1 = c_x - x$ is the time left before $x = c_x$ when time is moving forwards, and $b_1 = x$ is the time left before $x = 0$ when time is moving backwards.

If we now apply the update $x := c_x - x$, let a_2, b_2 be a_1, b_1 after we apply this update:

$$a_2 = c_x - (c_x - x), b_2 = (c_x - x)$$

$$\implies a_2 = x, b_2 = c_x - x$$

$$\implies a_2 = b_1, b_2 = a_1$$

\therefore this update swaps the time until an end point is reached, which is exactly what the flip operation does in the hourglass automata. Also, we remain in the $0 \leq x \leq c_x$ bound as expected ($c_x - x = c_x - [0, c_x] = [0, c_x]$).

This means we can let c_x be our only bound, and restrict time to progressing forwards. We also need to update the d map when we flip a clock x : $d(x) := -d(x)$.

We do not compare the clock value to anything above c_x , so we can just let the clock become greater than c_x . When a clock x is at its bounds and stopped progressing, the expected result of a flip on x is to let it start moving again towards the other bound, which would be c_x away. We can simulate this with a clock reset $x := 0$ and state change $d(x) := -d(x)$, then let the clock progress towards c_x . For this, we just have to check if $v(x) \geq c_x$, and execute the above steps in place of the $x := c_x - x$ update.

\therefore We can represent the hourglass automata's bounded clock and flip operation with timed automata, using the standard clock reset operation and the new update operation. \square

We will use timed automata extended with the $x := c_x - x$ update to show that the language emptiness problem on hourglass automata is decidable for two clocks or less. To prove decidability of the language emptiness problem on the hourglass automata, we will construct a finite untimed automaton that accepts untimed words that are equivalent to the timed words that are accepted by the corresponding timed automaton using the $x := c_x - x$ update. A finite untimed automata would have a finite number of states, so language emptiness would be decidable.

4.1 Clock Regions

In the standard timed automata, a finite untimed automata can be constructed using clock regions, which represent sets of clock valuations. From the reasoning given by Alur, Courcoubetis, and Dill [2], we can assume all comparisons will be between clocks and integers. Thus for every clock $x \in X$, we only care about its current integer value. The ordering of the fractional components is the other piece of information that is important so we know the order in which the integral components increase. This allowed the partition of the clock valuation space into equivalence classes. We will use a similar approach.

The original three constraints for two clock valuations, v and v' , to be equivalent ($v \cong v'$) are:

1. For all $x \in X$ either $v(x) \geq c_x$ and $v'(x) \geq c_x$, or $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$
2. For all $x, y \in X$ such that $v(x) \leq c_x$ and $v(y) \leq c_y$,
 $fr(v(x)) \leq fr(v(y))$ if and only if $fr(v'(x)) \leq fr(v'(y))$
3. For all $x \in X$ such that $v(x) \leq c_x$, $fr(v(x)) = 0$ if and only if $fr(v'(x)) = 0$

With the new update, $x := c_x - x$, we add one more constraint:

4. For all $x, y \in X$ such that $v(x) \leq c_x$ and $v(y) \leq c_y$,
 $fr(v(x)) + fr(v(y)) \leq 1$ if and only if $fr(v'(x)) + fr(v'(y)) \leq 1$ and
 $fr(v(x)) + fr(v(y)) \geq 1$ if and only if $fr(v'(x)) + fr(v'(y)) \geq 1$.

LEMMA 6 The fractional component constraints in 2 is mapped to the fractional component constraints in 4 and vice versa when the update $v'(x) := c_x - v(x)$ is made and the valuation of the clock isn't an integer. This means constraint 4 is required to preserve information when the flip operation is made.

Proof. From constraint 2 to constraint 4:

Let $x, y \in X$, $v(x) \leq c_x$, $v(x) \notin \mathbb{Z}$, $v(y) \leq c_y$, and $fr(v(x)) \leq fr(v(y))$.

Applying the update $v'(x) := c_x - v(x)$ gives us:

$$\begin{aligned} fr(v'(x)) \leq fr(v(y)) &\iff fr(c_x - v(x)) \leq fr(v(y)) \iff 1 - fr(v(x) - c_x) \leq fr(v(y)) \iff \\ 1 - fr(v(x)) \leq fr(v(y)) &\iff 1 \leq fr(v(x)) + fr(v(y)) \iff fr(v(x)) + fr(v(y)) \geq 1 \\ \therefore fr(v(x)) \leq fr(v(y)) &\rightarrow fr(v(x)) + fr(v(y)) \geq 1 \end{aligned}$$

Similarly, $fr(v(y)) \leq fr(v(x))$ updates to $fr(v(x)) + fr(v(y)) \leq 1$.

From constraint 4 to constraint 2:

Let $x, y \in X$ such that $v(x) \leq c_x$, $v(x) \notin \mathbb{Z}$, $v(y) \leq c_y$, and $fr(v(x)) + fr(v(y)) \leq 1$.

Applying the update $v'(x) := c_x - v(x)$ gives us:

$$\begin{aligned} fr(v'(x)) + fr(v(y)) \leq 1 &\iff fr(c_x - v(x)) + fr(v(y)) \leq 1 \iff fr(v(y)) \leq 1 - fr(c_x - v(x)) \iff \\ fr(v(y)) \leq fr(v(x) - c_x) &\iff fr(v(y)) \leq fr(v(x)) \\ \therefore fr(v(x)) + fr(v(y)) \leq 1 &\rightarrow fr(v(y)) \leq fr(v(x)) \end{aligned}$$

Similarly, $fr(v(x)) + fr(v(y)) \geq 1$ updates to $fr(v(x)) \leq fr(v(y))$.

Note that if $v(x) \in \mathbb{Z}$, then $fr(c_x - v(x)) = 0$, so the fractional constraints do not change and it is still consistent. \square

With only the original three constraints, \cong is known to be an equivalence relation.

LEMMA 7 \cong remains an equivalence relation with this new constraint.

Proof. We show that the three properties of an equivalence relation are maintained.

Reflexive: This is trivially true.

Symmetric:

Let $v, v' \in V$, and $\forall x, y \in X$ such that $v(x) \leq c_x$ and $v(y) \leq c_y$.

Either $v(x) = c_x \wedge v'(x) = c_x$ or $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$, so $v'(x) \leq c_x$ (similarly for y).

$\therefore v \cong v' \implies v' \cong v$.

Transitive:

Let $v, v', v'' \in V$, and: $\forall x, y \in X$ such that $v(x) \leq c_x$, $v'(x) \leq c_x$, $v(y) \leq c_y$ and $v'(y) \leq c_y$.

Either $v(x) = c_x \wedge v'(x) = c_x \wedge v''(x) = c_x$ or $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor = \lfloor v''(x) \rfloor$, so $v''(x) \leq c_x$ (similarly for y).

$\implies \forall x, y \in X$ such that $v(x) \leq c_x$ and $v(y) \leq c_y$

$f(v(x)) + fr(v(y)) \leq 1 \iff f(v'(x)) + fr(v'(y)) \leq 1 \iff f(v''(x)) + fr(v''(y)) \leq 1$

$f(v(x)) + fr(v(y)) \geq 1 \iff f(v'(x)) + fr(v'(y)) \geq 1 \iff f(v''(x)) + fr(v''(y)) \geq 1$

$\therefore v \cong v' \wedge v' \cong v'' \implies v \cong v''$.

The other constraints are unaffected, so \cong still defines an equivalence relation. \square

This means we have a partitioning of all possible clock valuations. Each equivalence class of \cong is called a *region*.

LEMMA 8 The number of regions induced by \cong is bounded by:

$$\prod_{x \in X} (2 \cdot (c_x + 1)) \cdot |X|! \cdot 2^{|X|-1} \cdot (|X| + 1)^{|X|} \cdot (|X| + 1)^{|X|}$$

Proof. A region can be described with five arrays:

1. For each clock $x \in X$, the interval in which x lies. Possible options for some clock x are:

$$\{[0, 0], (0, 1), [1, 1], \dots, (c_x - 1, c_x), [c_x, c_x], (c_x, \infty)\}$$

The number of possible ways to construct this array is $\prod_{x \in X} (2 \cdot (c_x + 1))$.

2. A permutation of $X_{v \leq c_x}$, $\beta : X \rightarrow \{1, 2, \dots, |X_{v \leq c_x}|\}$, giving the \leq ordering of the fractional components of the clocks with valuations that are still distinguishable.

The number of possible ways to construct this array is $|X_{v \leq c_x}|!$.

3. A boolean array of whether the fractional component of a clock is equal to the fractional component of the succeeding clock in the above permutation.

The number of possible ways to construct this array is $2^{|X_{v \leq c_x}|-1}$.

4. For each clock $x \in X_{v \leq c_x}$, the greatest integer $i \in \mathbb{N}_{\leq |X_{v \leq c_x}|}$ such that $fr(v(x)) + fr(v(x_i)) < 1$, where $\beta(x_i) = i$ or $v(x_0) = 0$ if no such i exists.

The number of possible ways to construct this array is $(|X_{v \leq c_x}| + 1)^{|X_{v \leq c_x}|}$.

5. For each clock $x \in X_{v \leq c_x}$, the greatest integer $i \in \mathbb{N}_{\leq |X_{v \leq c_x}|}$ such that $fr(v(x)) + fr(v(x_i)) \leq 1$, where $\beta(x_i) = i$ or $v(x_0) = 0$ if no such i exists.

The number of possible ways to construct this array is $(|X_{v \leq c_x}| + 1)^{|X_{v \leq c_x}|}$.

LEMMA 9 Every equivalence class of \cong can be represented by some five-tuple $\langle \alpha, \beta, \gamma, \zeta, \eta \rangle$, containing the arrays above.

Proof. Let $v, v' \in V$.

α consists of intervals of the form $[a, a]$ or $(a, a + 1)$ for $a \in \mathbb{N}_{< c_x}$, $[c_x, c_x]$, and (c_x, ∞) .

- $\alpha(x) = [a, a] \iff \lfloor v(x) \rfloor = a, \lfloor v'(x) \rfloor = a, fr(v(x)) = 0, fr(v'(x)) = 0$
 $\iff \lfloor v(x) \rfloor = \lfloor v'(x) \rfloor, fr(v(x)) = 0 \wedge fr(v'(x)) = 0$

- $\alpha(x) = (a, a) \iff \lfloor v(x) \rfloor = a, \lfloor v'(x) \rfloor = a, fr(v(x)) \neq 0, fr(v'(x)) \neq 0$
 $\iff \lfloor v(x) \rfloor = \lfloor v'(x) \rfloor, fr(v(x)) \neq 0 \wedge fr(v'(x)) \neq 0$
- $\alpha(x) = [c_x, c_x] \iff v(x) \geq c_x, v'(x) \geq c_x, fr(v(x)) = 0, fr(v'(x)) = 0$
 $\iff v(x) \geq c_x \wedge v'(x) \geq c_x, fr(v(x)) = 0 \wedge fr(v'(x)) = 0$
- $\alpha(x) = (c_x, \infty) \iff v(x) > c_x, v'(x) > c_x$
 $\iff v(x) > c_x \wedge v'(x) > c_x$

$\therefore \alpha$ satisfies constraints 1 and 3.

β and γ give us an \leq ordering of the fractional components of the clocks with valuations that are still distinguishable while giving us all sub-strings where the fractional components are equal.

- $\beta(x) = \beta(y) \iff v(x) \leq c_x \wedge v(y) \leq c_y \wedge v'(x) \leq c_x \wedge v'(y) \leq c_y \wedge fr(v(x)) = fr(v(y)) \wedge fr(v'(x)) = fr(v'(y))$
- $\beta(x) < \beta(y) \wedge (\bigwedge_{a \in \{a \in X \mid \beta(x) \leq \beta(a) \leq \beta(y)\}} (\gamma(a)))$
 $\iff v(x) \leq c_x \wedge v(y) \leq c_y \wedge v'(x) \leq c_x \wedge v'(y) \leq c_y \wedge fr(v(x)) = fr(v(y)) \wedge fr(v'(x)) = fr(v'(y))$
- $\beta(x) < \beta(y) \wedge \neg(\bigwedge_{a \in \{a \in X \mid \beta(x) \leq \beta(a) \leq \beta(y)\}} (\gamma(a))) \iff fr(v(x)) < fr(v(y)), fr(v'(x)) < fr(v'(y))$
 $\iff v(x) \leq c_x \wedge v(y) \leq c_y \wedge v'(x) \leq c_x \wedge v'(y) \leq c_y \wedge fr(v(x)) < fr(v(y)) \wedge fr(v'(x)) < fr(v'(y))$

$\therefore \beta$ and γ satisfy constraint 2.

ζ and η with β give us for each clock, the set of clocks that can be added so that sum is less than zero, equal to zero, and greater than zero.

- $\beta(y) \leq \zeta(x) \iff v(x) \leq c_x \wedge v'(x) \leq c_x \wedge v(y) \leq c_y \wedge v'(y) \leq c_y \wedge fr(v(x)) + fr(v(y)) < 1$
- $\eta(x) < \beta(y) \iff v(x) \leq c_x \wedge v'(x) \leq c_x \wedge v(y) \leq c_y \wedge v'(y) \leq c_y \wedge fr(v(x)) + fr(v(y)) > 1$
- $\zeta(x) < \beta(y) \leq \eta(x) \iff v(x) \leq c_x \wedge v'(x) \leq c_x \wedge v(y) \leq c_y \wedge v'(y) \leq c_y \wedge fr(v(x)) + fr(v(y)) = 1$

$\therefore \zeta$ and η with β satisfy constraint 4. \square

\therefore The number of clock regions is bounded by:

$$\prod_{x \in X} (2 \cdot (c_x + 1)) \cdot |X|! \cdot 2^{|X|-1} \cdot (|X| + 1)^{|X|} \cdot (|X| + 1)^{|X|}$$

\square

We next look at the elapsing of time, clock constraints, and clock updates.

LEMMA 10 Let v_1 and v_2 be two clock valuations, t be a non-negative integer, ϕ be a clock constraint, and $\lambda, \mu \subseteq X$ be a set of clocks. The following properties hold when the number of clocks is two or less:

1. $v_1 \cong v_2 \wedge t \in \mathbb{Z}^+ \implies v_1 + t \cong v_2 + t$
2. $v_1 \cong v_2 \implies \forall t_1 \in \mathbb{R}^+ \exists t_2 \in \mathbb{R}^+ [v_1 + t_1 \cong v_2 + t_2]$
3. $v_1 \cong v_2 \implies v_1 \text{ satisfies } \phi \iff v_2 \text{ satisfies } \phi$
4. $v_1 \cong v_2 \implies v_1[\lambda := 0] \cong v_2[\lambda := 0]$
5. $v_1 \cong v_2 \implies v_1[\mu := c_\mu - \mu] \cong v_2[\mu := c_\mu - \mu]$

Proof. (1)

Let $v_1 \cong v_2, t \in \mathbb{Z}^+$, then:

$$\begin{aligned} \forall x, y \in X \quad fr(v_1(x)) + fr(v_1(y)) \leq 1 &\iff fr(v_2(x)) + fr(v_2(y)) \leq 1 \text{ and} \\ fr(v_1(x)) + fr(v_1(y)) \geq 1 &\iff fr(v_2(x)) + fr(v_2(y)) \geq 1 \\ \implies \forall x, y \in X \quad fr(v_1(x) + t) + fr(v_1(y) + t) \leq 1 &\iff fr(v_2(x) + t) + fr(v_2(y) + t) \leq 1 \text{ and} \\ fr(v_1(x) + t) + fr(v_1(y) + t) \geq 1 &\iff fr(v_2(x) + t) + fr(v_2(y) + t) \geq 1 \\ \implies v_1 + t \cong v_2 + t &\end{aligned}$$

\square

Proof. (2)

When $t_1 = 0$: $t_2 = 0$

When $t_1 \in \mathbb{Z}^+$: $t_2 = t_1$ as above.

When $0 < t_1 < 1$: The existence of a t_2 is not guaranteed when there are more than two clocks (See 11).

We limit it to two clocks $X = \{x, y\}$ here, and $v_1 \cong v_2$:

1. Either $v_1(x) \geq c_x$ and $v_2(x) \geq c_x$, or $\lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor$ (similarly for y)
 2. $v_1(x) \leq c_x \wedge v_1(y) \leq c_y \implies fr(v_1(x)) \leq fr(v_1(y)) \iff fr(v_2(x)) \leq fr(v_2(y))$ and
 $fr(v_1(y)) \leq fr(v_1(x)) \iff fr(v_2(y)) \leq fr(v_2(x))$
 3. $v_1(x) \leq c_x \implies (fr(v_1(x)) = 0 \iff fr(v_2(x)) = 0)$ (similarly for y)
 4. $v_1(x) \leq c_x \wedge v_1(y) \leq c_y \implies fr(v_1(x)) + fr(v_1(y)) \leq 1 \iff fr(v_2(x)) + fr(v_2(y)) \leq 1$ and
 $fr(v_1(x)) + fr(v_1(y)) \geq 1 \iff fr(v_2(x)) + fr(v_2(y)) \geq 1$
- Case $v_1(x) + t_1 > c_x, v_1(y) + t_1 > c_y$: $t_2 > c_x - v_1(x) \wedge t_2 > c_y - v_1(y) \implies t_2$ is not bounded from above, and $t_2 = t_1 + 1$ will result in $v_2(x) + t_2 > c_x \wedge v_2(y) + t_2 > c_y$.
 - Case $v_1(x) + t_1 > c_x, v_1(y) + t_1 \leq c_y$: $c_x - v_2(x) < t_2 \wedge t_2 \leq c_y - v_2(y)$
 If $c_y - v_2(y) \geq 1$, then $t_2 = 1$ will work since $c_x - 1 < v_2(x) \leq c_x$ must be true.
 If $0 < c_y - v_2(y) < 1 \implies \lfloor v_2(y) \rfloor = c_y - 1 \implies \lfloor v_1(y) \rfloor = c_y - 1 \implies$
 $fr(v_1(y)) < fr(v_1(x)) \implies fr(v_2(y)) < fr(v_2(x)) \implies$
 $t_2 = 1 - fr(v_2(y))$ will work since $1 - fr(v_2(x)) < t_2 \leq 1 - fr(v_2(y))$ is the range t_2 must be in and this range is not empty.
 $c_y - v_2(y) = 0$ cannot be true since that would mean $v_2(y) = c_y \iff v_1(1) = c_y \iff t_1 = 0$, which is a contradiction since $0 < t_1 < 1$.
 - Case $v_1(x) + t_1 \leq c_x, v_1(y) + t_1 > c_y$: We can find a t_2 using the same reasoning as above.
 - Case $v_1(x) + t_1 \leq c_x, v_1(y) + t_1 \leq c_y, fr(v(y)) \leq fr(v(x))$: (The $fr(v(x)) \leq fr(v(y))$ case is similar)

Let's assume neither integral components change when t_1 is added, then constraints 1 to 3 only require that t_2 doesn't change either integral components as well.

The regions we get from the fourth constraint:

1. $fr(v(x)) + fr(v(y)) < 1 \wedge fr(v(y)) = 0$
2. $fr(v(x)) + fr(v(y)) < 1$
3. $fr(v(x)) + fr(v(y)) = 1 \wedge fr(v(x)) \neq 0$
4. $fr(v(x)) + fr(v(y)) > 1 \wedge fr(v(x)) \neq 0$

From any point in these regions, you can reach a point in the following region:

- 1 \rightarrow 2 Let $t_2 = \varepsilon$, where $0 < \varepsilon < 1 - (fr(v(x)) + fr(v(y)))$
- 2 \rightarrow 3 Let $t_2 = 1 - (fr(v(x)) + fr(v(y)))$
- 3 \rightarrow 4 Let $t_2 = \varepsilon$, where $\varepsilon > 0$, $\lfloor v(x) \rfloor = \lfloor v(x) + \varepsilon \rfloor$,
 $\lfloor v(y) \rfloor = \lfloor v(y) + \varepsilon \rfloor$
- 4 \rightarrow 1 Let $t_2 = \min(1 - fr(v(x)), 1 - fr(v(y)))$

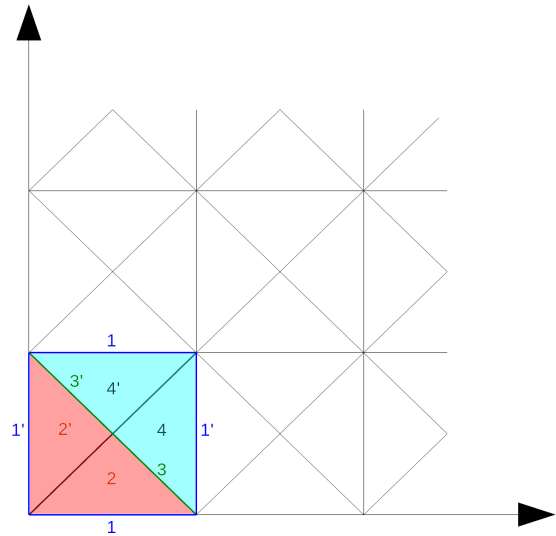


Figure 1: The four region types

After the fourth region, $fr(v(x)) = 0$ and the integral component of x is incremented before starting again at the first region, but with x and y variable positions swapped. The transition from 4 to 1 also aligns with the integral component increment and fractional component ordering change, so they are taken into account.

When $t_1 > 1, t_1 \notin \mathbb{Z}^+$:

Let $t'_1 = t_1 - \lfloor t_1 \rfloor \implies 0 < t'_1 < 1$, and

$\lfloor t_1 \rfloor \in \mathbb{Z}^+ \iff v_1 + \lfloor t_1 \rfloor \cong v_2 + \lfloor t_1 \rfloor$, so if we let $v'_1 = v_1 + \lfloor t_1 \rfloor, v'_2 = v_2 + \lfloor t_1 \rfloor$,

then we just need to find t'_2 , where $v'_1 + t'_1 \cong v'_2 + t'_2$. \square

Proof. (3) Let $x, y \in X$

$v_1 \cong v_2$ and v_1 satisfies ϕ , where ϕ compares clocks to integers:

$\forall x \in X, a_x, b_x \in \mathbb{Z}_{\leq c_x}^+, a_x \leq v(x) \leq b_x$

$\implies a_x, b_x, a_y, b_y \in \mathbb{Z}_{\leq c_x}^+, a_x \leq v_1(x) \leq b_x \wedge a_y \leq v_1(y) \leq b_y \wedge$

$\lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor \wedge \lfloor v_1(y) \rfloor = \lfloor v_2(y) \rfloor \wedge$

$fr(v_1(x)) = 0 \iff fr(v_2(x)) = 0 \wedge fr(v_1(y)) = 0 \iff fr(v_2(y)) = 0$

$\implies a_x \leq v_2(x) \leq b_x \wedge a_y \leq v_2(y) \leq b_y$

$\implies v_2$ satisfies ϕ , and it is unaffected by the new constraint. \square

Proof. (4) Let $x, y \in X, \lambda \subseteq X$

$v_1 \cong v_2 \implies$

$fr(v_1(x)) + fr(v_1(y)) \leq 1 \iff fr(v_2(x)) + fr(v_2(y)) \leq 1 \wedge$

$fr(v_1(x)) + fr(v_1(y)) \geq 1 \iff fr(v_2(x)) + fr(v_2(y)) \geq 1$

$v_1[\lambda := 0], (x \in \lambda \vee y \in \lambda) \implies fr(v_1(x)) + fr(v_1(y)) \leq 1$

$v_2[\lambda := 0], (x \in \lambda \vee y \in \lambda) \implies fr(v_2(x)) + fr(v_2(y)) \leq 1$

$\therefore v_1[\lambda := 0], v_2[\lambda := 0], (x \in \lambda \vee y \in \lambda) \implies fr(v_1(x)) + fr(v_1(y)) \leq 1, fr(v_2(x)) + fr(v_2(y)) \leq 1$

$\implies v_1[\lambda := 0] \cong v_2[\lambda := 0]$ since the other constraints are already known to be satisfied. \square

Proof. (5) Let $x \in \mu, \mu \subseteq X$

If $v(x) \geq c_x$, then the *flip* operation is converted to a reset operation as explained in Section 4, so we can assume $v(x) < c_x$.

$v_1 \cong v_2 \implies \lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor \wedge (fr(v_1(x)) = 0 \iff fr(v_2(x)) = 0)$

$fr(v_1(x)) = 0 \implies \lfloor c_x - v_1(x) \rfloor = c_x - v_1(x) = c_x - \lfloor v_1(x) \rfloor, fr(c_x - v_1(x)) = 0$

$fr(v_2(x)) = 0 \implies \lfloor c_x - v_2(x) \rfloor = c_x - v_2(x) = c_x - \lfloor v_2(x) \rfloor, fr(c_x - v_2(x)) = 0$

$fr(v_1(x)) \neq 0 \implies \lfloor c_x - v_1(x) \rfloor = c_x - \lceil v_1(x) \rceil = c_x - (\lfloor v_1(x) \rfloor + 1), fr(c_x - v_1(x)) \neq 0$

$fr(v_2(x)) \neq 0 \implies \lfloor c_x - v_2(x) \rfloor = c_x - \lceil v_2(x) \rceil = c_x - (\lfloor v_2(x) \rfloor + 1), fr(c_x - v_2(x)) \neq 0$

$\therefore \lfloor c_x - v_1(x) \rfloor = \lfloor c_x - v_2(x) \rfloor \wedge (fr(c_x - v_1(x)) = 0 \iff fr(c_x - v_2(x)) = 0)$ is true, so constraints 1 and 3 are maintained after the flip operation.

Let $x, y \in X, \mu \subseteq X$

From Lemma 6, we know that if $x \in \mu, y \notin \mu, v(x) \notin \mathbb{Z}$ or $x, y \in \mu, v(x) \notin \mathbb{Z}, v(y) \in \mathbb{Z}$:

$fr(v(x)) \leq fr(v(y)) \rightarrow fr(v(x)) + fr(v(y)) \geq 1,$

$fr(v(y)) \leq fr(v(x)) \rightarrow fr(v(x)) + fr(v(y)) \leq 1,$

$fr(v(x)) + fr(v(y)) \leq 1 \rightarrow fr(v(y)) \leq fr(v(x)),$

$fr(v(x)) + fr(v(y)) \geq 1 \rightarrow fr(v(x)) \leq fr(v(y))$

and $x, y \in \mu, v(x), v(y) \notin \mathbb{Z} \implies$

$fr(v(x)) \leq fr(v(y)) \rightarrow fr(v(x)) + fr(v(y)) \geq 1 \rightarrow fr(v(y)) + fr(v(x)) \geq 1 \rightarrow fr(v(y)) \leq fr(v(x)),$

$fr(v(y)) \leq fr(v(x)) \rightarrow fr(v(x)) + fr(v(y)) \leq 1 \rightarrow fr(v(y)) + fr(v(x)) \leq 1 \rightarrow fr(v(x)) \leq fr(v(y))$,
 $fr(v(x)) + fr(v(y)) \leq 1 \rightarrow fr(v(y)) \leq fr(v(x)) \rightarrow fr(v(x)) \leq fr(v(y)) \rightarrow fr(v(x)) + fr(v(y)) \geq 1$,
 $fr(v(x)) + fr(v(y)) \geq 1 \rightarrow fr(v(x)) \leq fr(v(y)) \rightarrow fr(v(y)) \leq fr(v(x)) \rightarrow fr(v(x)) + fr(v(y)) \leq 1$
 and $x \in \mu, v(x) \in \mathbb{Z}$ or $x, y \in \mu, v(x), v(y) \in \mathbb{Z} \implies$ no fractional component constraints change with the flip operation.

From the above mappings, we can see that all fractional component constraints are being mapped consistently to a fractional component constraint.

\therefore constraints 2 and 4 are maintained after the flip operation. \square

LEMMA 11 The progression of time is where this clock update faces undecidability issues when there are more than two clocks. The condition $v_1 \cong v_2 \implies \forall t_1 \in \mathbb{R}^+ \exists t_2 \in \mathbb{R}^+ [v_1 + t_1 \cong v_2 + t_2]$ from above is not met with three clocks.

Proof. We show an example with three clocks:

Let $x, y, z \in X$ be the three clocks, and $c_x = 2, c_y = 2, c_z = 2, v_1(x) = 0.4, v_1(y) = 0.4, v_1(z) = 0.8, v_2(x) = 0.1, v_2(y) = 0.1, v_2(z) = 0.95$

We show that $v_1 \cong v_2$:

1. $\lfloor v_1(x) \rfloor = 0, \lfloor v_2(x) \rfloor = 0 \implies \lfloor v_1(x) \rfloor = \lfloor v_2(x) \rfloor$
 $\lfloor v_1(y) \rfloor = 0, \lfloor v_2(y) \rfloor = 0 \implies \lfloor v_1(y) \rfloor = \lfloor v_2(y) \rfloor$
 $\lfloor v_1(z) \rfloor = 0, \lfloor v_2(z) \rfloor = 0 \implies \lfloor v_1(z) \rfloor = \lfloor v_2(z) \rfloor$
2. $fr(v_1(x)) = 0.4, fr(v_1(y)) = 0.4, fr(v_2(x)) = 0.1, fr(v_2(y)) = 0.1 \implies$
 $fr(v_1(x)) \leq fr(v_1(y))$ and $fr(v_2(x)) \leq fr(v_2(y))$
 Also true when we swap x and y.
 $fr(v_1(x)) = 0.4, fr(v_1(z)) = 0.8, fr(v_2(x)) = 0.1, fr(v_2(z)) = 0.95 \implies$
 $fr(v_1(x)) \leq fr(v_1(z))$ and $fr(v_2(x)) \leq fr(v_2(z))$
 When swapped, both constraints fail as expected.
 $fr(v_1(y)) = 0.4, fr(v_1(z)) = 0.8, fr(v_2(y)) = 0.1, fr(v_2(z)) = 0.95 \implies$
 $fr(v_1(y)) \leq fr(v_1(z))$ and $fr(v_2(y)) \leq fr(v_2(z))$
 When swapped, both constraints fail as expected.
3. None have fractional components of zero.
4. $fr(v_1(x)) = 0.4, fr(v_1(y)) = 0.4, fr(v_2(x)) = 0.1, fr(v_2(y)) = 0.1 \implies$
 $fr(v_1(x)) + fr(v_1(y)) < 1$ and $fr(v_2(x)) + fr(v_2(y)) < 1$
 Also true when we swap x and y.
 $fr(v_1(x)) = 0.4, fr(v_1(z)) = 0.8, fr(v_2(x)) = 0.1, fr(v_2(z)) = 0.95 \implies$
 $fr(v_1(x)) + fr(v_1(z)) > 1$ and $fr(v_2(x)) + fr(v_2(z)) > 1$
 Also true when we swap x and z.
 $fr(v_1(y)) = 0.4, fr(v_1(z)) = 0.8, fr(v_2(y)) = 0.1, fr(v_2(z)) = 0.95 \implies$
 $fr(v_1(y)) + fr(v_1(z)) > 1$ and $fr(v_2(y)) + fr(v_2(z)) > 1$
 Also true when we swap y and z.

$\therefore v_1 \cong v_2$, but if we choose $t_1 = 0.1 \implies (v_1 + t_1)(x) = 0.5, (v_1 + t_1)(y) = 0.5, (v_1 + t_1)(z) = 0.9$:

1. $\lfloor (v_2 + t_2)(x) \rfloor = 0, \lfloor (v_2 + t_2)(y) \rfloor = 0, \lfloor (v_2 + t_2)(z) \rfloor = 0 \implies t_2 \leq 0.9, t_2 \leq 0.9, t_2 \leq 0.05$
 $t_2 < 0.05$
2. No fractional ordering will change unless there is an integral component change, so this has the same restriction of $t_2 < 0.05$.

3. $t_2 < 0.05$ for the same reason as above.
4. $fr((v_1 + t_1)(x)) + fr((v_1 + t_1)(y)) = 1 \implies fr((v_2 + t_2)(x)) + fr((v_2 + t_2)(y)) = 1$
 $fr((v_1 + t_1)(x)) + fr((v_1 + t_1)(z)) > 1 \implies fr((v_2 + t_2)(x)) + fr((v_2 + t_2)(y)) > 1$
 $fr((v_1 + t_1)(y)) + fr((v_1 + t_1)(z)) > 1 \implies fr((v_2 + t_2)(x)) + fr((v_2 + t_2)(y)) > 1$
 $\implies t_2 = 0.4, t_2 < 0.475, t_2 < 0.475$

We have two conflicting requirements where $t_2 < 0.05$ but $t_2 = 0.4$. This means the interval for z and the ordering on the fractional components must change before the $x + y$ sum constraint can be fulfilled.

\therefore There is no t_2 such that $v_1 + t_1 \cong v_2 + t_2$. \square

We now extend our equivalence relation \cong over clock valuations to an equivalence relation over the state space of timed automata by taking the cross product of the locations, clock valuations and the rate of change map, and require that equivalent states have the same location, clock region and rate of change map.

So let $s, s' \in S$, $v, v' \in V$ and $d, d' \in D$, then $(s, [v], d) \cong (s', [v'], d') \iff s = s' \wedge [v] = [v'] \wedge d = d'$.

We will call this the *region graph*.

LEMMA 12 For every timed action in the timed automata, there exists a corresponding transition in the region graph.

Proof. Let $v_1 \cong v_2 \wedge (s, v_1) \Rightarrow_a (s', v'_1)$. For simplicity, we will drop the encoding of the rate of change map since they can also be encoded in s .

The transition $\langle s, a, \phi, \lambda, \mu, s' \rangle$ that changes the state from (s, v_1) to (s', v'_1) corresponds to two transitions of the timed automaton:

- a delay transition $(s, v_1) \rightarrow_{t_1} (s, v_1 + t_1)$ for some $t_1 \geq 0$, followed by
- an action transition $(s, v_1 + t_1) \rightarrow_a (s, v'_1) : v_1 + t_1$ satisfies ϕ and $v'_1 = (v_1 + t_1)[\lambda := 0, \mu := c_\mu - \mu]$.

Delay transition:

$$v_1 \cong v_2 \implies \forall t_1 \in \mathbb{R}^+ \exists t_2 \in \mathbb{R}^+ [v_1 + t_1 \cong v_2 + t_2]$$

$$\therefore \exists t_2 : (s, v_2) \rightarrow_{t_2} (s, v_2 + t_2) \text{ and } (s, v_1 + t_1) \cong (s, v_2 + t_2)$$

Action transition:

$$(v_1 + t_1) \cong (v_2 + t_2) \wedge (v_1 + t_1) \text{ satisfies } \phi \implies (v_2 + t_2) \text{ satisfies } \phi$$

$$(v_1 + t_1) \cong (v_2 + t_2) \implies (v_1 + t_1)[\lambda := 0] \cong (v_2 + t_2)[\lambda := 0]$$

$$(v_1 + t_1)[\lambda := 0] \cong (v_2 + t_2)[\lambda := 0] \implies (v_1 + t_1)[\lambda := 0, \mu := c_\mu - \mu] \cong (v_2 + t_2)[\lambda := 0, \mu := c_\mu - \mu]$$

$$\therefore \exists v'_2 = (v_2 + t_2)[\lambda := 0, \mu := c_\mu - \mu] : v'_1 \cong v'_2 \wedge (s, v_2) \Rightarrow_a (s', v'_2)$$

\therefore We have transitions in the region graph that correspond to transitions of the timed automaton, and the behaviour is consistent. \square

Lastly, the initial states in the region graph have the form:

$$(s_i, [v], d), \text{ where } s_i \in S_i, \text{ and } \forall x \in X, v(x) = 0 \wedge d(x) = 1$$

The final states in the region graph have the form:

$$(s_f, [v], d), \text{ where } s \in S_f$$

LEMMA 13 The constructed region graph will accept exactly the set of words equivalent to the words accepted by the corresponding timed automaton.

The proof for this will be the same as the proof given for the standard timed automata [3].

THEOREM 14 The language emptiness problem on hourglass automata with two or fewer clocks is decidable.

5 Stopping Time

LEMMA 15 With two hourglass clocks, the stopping of time does not cause decidability issues.

Hourglass clocks are never compared to one another, and they are bounded to a range. This means we only need to consider the progression of time.

In the 2-clock region diagram shown in Figure 1, we can see that a horizontal or vertical time progression can be made to work by introducing an extra constraint:

$$\forall x \in X \text{ such that } v(x) \leq c_x \text{ } fr(v(x)) \leq 0.5 \iff fr(v'(x)) \leq 0.5 \text{ and } fr(v(x)) \geq 0.5 \iff fr(v'(x)) \geq 0.5.$$

This cuts the second and fourth region in Figure 1 into two halves. Proving that \cong remains an equivalence relation with this extra condition requires few modifications to the original ones since this new condition is an extended version of condition 4, but we let clocks x and y to be the same clock.

The two properties to prove are:

1. $v_1 \cong v_2 \wedge t \in \mathbb{Z}^+ \implies v_1 + t \cong v_2 + t$
2. $v_1 \cong v_2 \implies \forall t_1 \in \mathbb{R}^+ \exists t_2 \in \mathbb{R}^+ [v_1 + t_1 \cong v_2 + t_2]$

Proof. (1) This is unaffected from the proof given where no clocks are stopped. \square

Proof. (2) We can see from Figure 1 that when we cut the second and fourth regions, the region transitions are fixed when we allow time to progress. Note that this does not work for three clocks since when you have one stopped clock with one of the other two clocks having a higher fractional component and the last clock having a lower fractional component than the stopped clock, you cannot determine whether the lower will cross the stopped clock or the higher will reach the next integer first. \square

6 Conclusion and Future Work

In this paper, we introduced the hourglass automata and its ability to let clocks go backwards within some bounded range. To be able to reason and study this class of automata, we first introduced an extension to the standard timed automata which has a new update, $x := c - x$, where x is a clock, $c \in \mathbb{N}_{\leq c_x}$, and c_x is the greatest integer constant that clock x is compared against, and a rate of change map $D : X \rightarrow \{-1, -0, 0, 1\}$ is added to the state space. We then showed that this extended timed automata is capable of expressing the operations of the hourglass automata by showing a translation of the flip operation on bounded clocks. As a result, this allowed us to examine the language emptiness problem for this class of automata.

For the hourglass automata, we limited the clock update to the form $x := c_x - x$ since this was all that was necessary. From there, we prove the decidability of the language emptiness problem on this class of automata when two or fewer clocks are involved, using the same approach used for proving decidability of the standard timed automata [3]. This result shows that for two hourglass clocks, we can reduce the system to a finite graph.

The next step would be to investigate the possibility of having more than two clocks. The timed automata that we currently have defined is capable of more than the hourglass automata definition, and maintains a lot of information, which is not necessary, so it is possible that we could use another model to represent the hourglass automata such that language emptiness verification is decidable with more than two clocks. Some important notes here would be removing all unnecessary constraints that are not the boundaries, and possibly introduce clock regions which are not square with edges on the integer components. This may be possible since we don't compare to integer constants other than the bounds.

The ability for hourglasses to be placed on their side, allowing the stopping of clocks, will also be an interesting area to investigate when more than two clocks are introduced. In general, the ability to stop time allows timed automata to become as expressive as linear hybrid automata [7], thus introducing undecidability issues, but there has been other classes of timed automata that have a limited ability to stop time like the interrupt timed automata [11]. The interrupt timed automata would have some overlap with the hourglass automata in what it can express.

Additionally, the decidability proof for the update given in this paper can apply for any $c \in \mathbb{N}_{\leq c_x}$ as long as $c - x \geq 0$, otherwise the clock values become invalid, so this extended timed automata could be developed further.

References

- [1] R. Alur and D. Dill. Automata for modeling real-time systems. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 322–335. Springer, 1990.doi:10.1007/BFb0032042
- [2] R. Alur, C. Courcoubetis, and D. L. Dill. Model-checking for real-time systems. In *Proceedings of the 5th Annual Symposium on Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, 1990.doi:10.1109/LICS.1990.113766
- [3] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.doi:10.1016/0304-3975(94)90010-8
- [4] D. Wells. *The Penguin Book of Curious and Interesting Puzzles*. Penguin Books, 1992.
- [5] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What’s Decidable About Hybrid Automata? In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing (STOC)*, pages 373–382. ACM, 1995.doi:10.1145/225058.225162
- [6] R. Alur, C. Courcoubetis, I. N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.doi:10.1016/0304-3975(94)00202-T
- [7] F. Cassez, and K. G. Larsen. The Impressive Power of Stopwatches. In *Proceedings of CONCUR 2000: Concurrency Theory*, pages 128–152. Springer, 2000.doi:10.1007/3-540-44618-4_12
- [8] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. In *International Journal on Software Tools for Technology Transfer*, 1(2):134–152. Springer, 1997.doi:10.1007/s100090050010
- [9] P. Bouyer, C. Dufourd, E. Fleury and A. Petit. Updatable Timed Automata. *Theoretical Computer Science*, 321(2-3):291–345, 2004.doi:10.1016/j.tcs.2004.04.003
- [10] T. Brihaye, L. Doyen, G. Geeraerts, J. Ouaknine, J.-F. Raskin, J. Worrell. On Reachability for Hybrid Automata over Bounded Time. In *International Colloquium on Automata, Languages and Programming (ICALP)*, (2):416–427. Springer, 2011.doi:10.1007/978-3-642-22012-8_33
- [11] B. Bérard, S. Haddad and M. Sassolas. Interrupt Timed Automata: verification and expressiveness. In *Formal Methods in System Design*, 40(1):41–87. Springer, 2012.doi:10.1007/s10703-011-0140-2
- [12] E. Asarin, V. Mysore, A. Pnueli, G. Schneider. Low dimensional hybrid systems - decidable, undecidable, don’t know. In *Information and Computation*, 211:138–159. Academic Press, 2012.doi:10.1016/j.ic.2011.11.006